EHANK Code Review

William Chen and Emily Martell

DSGE Interns 2018

August 6, 2018

Outline

- Overview
- Models
- Solve
- Estimate
- Other files

Overview

Instantiating a model object

- m = KrusellSmith() or m = OneAssetHANK()
- Sets up a KrusellSmith model object with parameters, settings, endogenous states, exogenous shocks
- Computes and stores steady states

Solving the model

- solve(m)
- Uses parameters, settings, steady states
- Computes canonical form: approximates derivatives of value function, distribution, etc.
- Reduces canonical form matrices: Krylov reduction and value function reduction
- Runs continuous time gensys over reduced matrices
- Returns transition equation matrices

Estimating the model

- estimate(m,data)
- Uses parameters, settings, steady states
- Solves model, resulting in transition equation matrices
- In DSGE: Finds posterior mode, computes proposal distribution for MCMC, runs SMC
- Yields parameter posterior distribution (stores in saveroot)

Models

models\

- solve_hjb.jl
- solve_kfe.jl
- krusell_smith\
- one_asset_hank\

models\krusell_smith\

- augment_states.jl
- data\
  - generate_data.jl
- eqcond.jl
- krusell_smith.jl
- measurement.jl
- observables.jl
- subspecs.jl

models\one_asset_hank\

- augment_states.jl
- data\
  - generate_data.jl
- eqcond.jl
- helpers.jl
- one_asset_hank.jl
- measurement.jl
- observables.jl
- subspecs.jl

Major Changes

1. Handling of model indices
2. Storage of steady state parameters
3. Extension of Setting
4. Computation of steady state

1. Model Indices

- Location: krusell_smith.jl, one_asset_hank.jl
- Store endogenous_states indices as vectors of Ints (rather than just Ints)
- Why? value function/cross-sectional distribution are described by vectors

2. Steady State Parameters

- Location: parameters_hank.jl
- Created new types SteadyStateParameterVector, SteadyStateParameterArray
- Why?
  1. Some steady state parameters, like the steady state value functions, are vectors. Others, like the ones which indicate when forward/backward differentiation occurs, are matrices.
  2. SteadyStateParameter doesn't allow for storage of vectors and methods for updating SteadyStateParameter are not adapted for updating vector values

3. Settings

- Location: krusell_smith.jl, one_asset_hank.jl
- Need place to store info on ....
  1. chosen state space grid discretization
  2. number of jump variables, aggregate state variables, static relations, expectation errors, and shocks.
  3. variables governing the accuracy of the steady-state approximation
  4. variables governing application of reduction
  5. filtering method (e.g. are we tracking the last lag- more on this in theory portion later today)
  6. SMC

More on Settings

- Most grid-related settings don't have economic meaning (e.g. fineness of wealth grid), but some do (e.g. idiosyncratic income Markov chain has values 0 and 1, corresponding to unemployment and employment)
- Some settings are required. We made error messages in later parts of the code that will tell the user when a certain setting is needed for a certain functionality (e.g. reduction).

Mandatory Settings

- n_jump_vars: Number of jump/control variables
- n_state_vars: Number of aggregate state variables (taken as given by agent)
- n_state_vars_unreduce: Number of state vars not to be reduced, usually zero
- n_static_conditions: Number of static conditions
- n_shocks: Number of aggregate shocks
- n_expectation_errors: Expectational errors for canonical form, usually the value function
- n_vars: n_jump_vars + n_state_vars + n_static_conditions

Mandatory Settings for Reduction

- reduce_state_vars: Boolean for whether we reduce state variables like g_t
- reduce_v: Boolean for whether we do spline reduction of value function
- krylov_dim: Dimension of Krylov subspace
- n_prior: How many non-wealth dimensions are accounted for by spline basis
- n_post: How many non-wealth dimensions should be accounted for by spline basis
- knots_dict: Dictionary holding knot points for each dimension used for spline basis
- spline_grid: grid of knot point locations

4. Steady State Computation

- Location: krusell_smith.jl, one_asset_hank.jl
- Algorithm
  1. Read in all parameters and relevant settings
  2. Initialize guesses and storage variables (e.g. matrices for flow utility)
  3. Iterate the following steps until some market-clearing condition is met
     1. Solve for HJB given prices and other quantities
     2. Solve KFE given value function and prices
     3. Compute market-clearing conditions. Check if steady state is found

Auxiliary Steady-State Functions

- solve_kfe.jl:
  1. Holds a variety of solvers that return a stacked distribution vector.
  2. Why? given A matrix from HJB, solving KFE is model-invariant. Makes sense to collect this in a function
  3. Contains both exact (via left divide) and iterative methods for computing g_t.
  4. Exact method generalizes to any number of state space grid dims.
  5. Iterative method may not generalize (we don't know as we didn't test this)
  6. Many redundant function definitions b/c some arrays may be complex, vectors, matrices, etc.

Auxiliary Steady-State Functions

- solve_hjb.jl:
  1. Why? solving HJB has several steps which may be model invariant, makes sense to collect this in a function
  2. Holds 3 main functions: upwind_value_function, upwind_matrix, solve_hjb
  3. Has an upwind function: wrapper script for upwind_value_function and upwind_matrix
  4. upwind_value_function: determines which parts of the state space uses forward difference, backward difference, etc.
  5. upwind_matrix: Constructs A matrix for HJB.
  6. solve_hjb: Exact method (left-divide) for solving HJB (we don't have an iterative method yet, though this should be a straightforward extension)

Why upwind_value_function?

- Location: solve_hjb.jl
- Exact upwind scheme only depends on value function finite differences -> model agnostic
- Inexact method may not generalize beyond CRRA utility and CRRA utility w/labor disutility b/c how we decide which points in state space grid get a forward difference, backward difference etc.

Why upwind_matrix?

- Location: solve_hjb.jl
- Upwind matrix assumes wealth is a state variable, but this should be true for most HANK models
- Creates part of A matrix that applies to the drift of wealth in value function
- Given A_switch, which summarizes idiosyncratic shocks, procedure for creating A is model invariant

Why solve_hjb?

- Location: solve_hjb.jl
- Left-divide exact method is model invariant as long as discount rate is constant

Algorithm for eqcond

- Location: krusell_smith\eqcond.jl, one_asset_hank\eqcond.jl
- Need to know number of
  1. total variables (jump + states + static)
  2. expectation errors (usually jump vars)
  3. shocks
- Grab all relevant parameters/settings
- Create a vector holding steady state values, time derivatives, expectational errors, and shocks
- Compute one iteration of the HJB solution method
- Compute residuals of jump/state/static variables based off that iteration
- Apply automatic differentiation to the residuals
- Grab the relevant gensys matrices
- Didn't do much generalization here, but could conceivably make helper functions for hjb and kfe_residuals

Solve

solve\

- gensysct.jl
- reduction.jl
- solve.jl

solve function

- Location: solve.jl
- Computes equilibrium condition matrices (eqcond: in krusell_smith\eqcond.jl, one_asset_hank\eqcond.jl)
- Check for whether mandatory settings are populated (can be turned off for speed)
- Reduction: Krylov reduction (krylov_reduction) and value function reduction (valuef_reduction)
- Gensys (gensysct)
- Returns T, C, R, inverse_basis, basis
  - inverse_basis moves matrices from reduced basis to full dimension, basis does opposite
  - inverse_basis $\times$ basis $= I$ but basis $\times$ inverse_basis $\neq I$

reduction

- Location: reduction.jl
- Main functions: krylov_reduction and valuef_reduction
- Auxiliary functions: deflated_block_arnoldi, change_basis, spline_basis, extend_to_nd, projection_for_subset, solve_static_conditions
- Function we probably want to remove: oneDquad_spline (spline_basis is the generalization of this function; we no longer call oneDquad_spline)

krylov_reduction

1. Location: reduction.jl
2. Slice up Γ_1 (See Why Inequality Matters for details)
3. Compute Krylov subspace
4. Build state space reduction transform, inverse transform
5. Perform change of basis (change_basis)
6. Return canonical form matrices

- Saves a number of state variables
- Note that static conditions are dropped from state vector after Krylov reduction (as these depend entirely on jump and aggregate state variables)

valuef_reduction

1. Location: reduction.jl
2. Create spline basis (spline_basis)
3. Extend spline basis to other dimension which we did not use a spline approximation (extend_to_nd)
4. Create projection matrix that projects value function onto spline basis (projection_for_subset)
5. Reduce the decision vector (change_basis)
6. Return canonical form matrices

- Saves a number of jump variables

gensysct

- Location: gensysct.jl
- Four versions
  1. Γ_0 need not be identity, in place Schur factorization
  2. Γ_0 need not be identity, not in place Schur factorization
  3. Γ_0 is identity, in place Schur factorization
  4. Γ_0 is identity, not in place Schur factorization
- Initial speed tests suggest that the in place gensysct may actually be slower (see timing in docs)

Estimate

estimate\

- filter_hank.jl
- hessizero_hank.jl
- transform_transition_matrices.jl
- ct_filter\
  - block_kalman_filter.jl
  - ct_block_kalman_filter.jl
  - ct_kalman_filter.jl

Modified Files

- filter_hank.jl: Returns output of Kalman filter with state transition matrices transformed to accommodate simulated subintervals (calls transform_transition_matrices)
- hessizero_hank.jl: allow for just one free parameter, as both OneAssetHANK and KrusellSmith have one shock (original code assumed more than 1)
- transform_transition_matrices.jl: Transform state transition matrices if estimating by simulating subintervals

Future Directions

- Since we don't estimate in any other way, we assume that the user will simulate subintervals via an Euler-Maruyama scheme.
- If we add other schemes for CT Kalman filtering, then we need to alter filter_hank.jl to not make this assumption
- track_lag: It seemed sensible not to augment states because it's only necessary depending on how we implement the measurement equation. So we leave track_lag as a setting and make transform_transition_matrices work w/ and w/out this setting.
- If we choose to augment states (we currently don't), it will require refactoring transform_transition_matrices (and maybe other functions) since solve will return a T matrix which is twice as large, is half zero, and has an identity in the upper right block.

ct_filters

- block_kalman_filter: Implements a Kalman Filter that speeds up multiplication by taking advantage of the block structure of matrices common to DSGE models. Only implemented one case of the referenced paper
- ct_block_kalman_filter: Incomplete Kalman Filter that exploits the structure of the TTT and RRR matrices when using simulated subintervals.
- ct_kalman_filter: Kalman filter with method to allow for ODE integration scheme for prediction step

Other Files

src\

- abstractdsgemodel_hank.jl
- defaults_hank.jl
- EHANK.jl
- parameters_hank.jl
- statespace_hank.jl
- util_hank.jl
- models\
- solve\
- estimate\
- runfiles\

Source Files

- parameters_hank.jl: Implement SteadyStateParameterVector and SteadyStateParameterArray
- abstractdsgemodel_hank.jl: Implement assignment via <= and setindex! for new parameter types
- statespace_hank.jl: Update compute_system since we track states in the reduced basis but setting up measurement equation is easier in the full state space.
- util_hank.jl: Perform inequality with complex numbers

Future Directions

- Should naming convention change? (e.g. files end with _hank)
- Can we suppress warnings issued when loading package?
- parameters_hank
  1. In the past, we were unable to add or insert a ScaledParameter type to a sparse matrix. There may also be other functionalities that cannot be used because matrix/scalar operations with DSGE Parameter types are not defined for everything that could come up (e.g. will they work with .* and left divide?

Run Files

- These just run SMC. These are for testing purposes

test and test_outputs

- test holds test files for krusell_smith.jl, one_asset_hank.jl, gensysct.jl, reduction.jl, and solve.jl
- test_outputs: Holds saved data that is used by test files.

Doc files

- G0_identity holds the explanation for why Γ_0 is the identity
- kalman_filter holds experimentation files assessing the accuracy of MLE with different Kalman Filters and the run times.
- measurement holds
  1. the explanation for how we approximate the integral used to convert flow variables to stock variables in the measurement equation
  2. experimentation files for how many terms in the power series expansion are needed to well approximate the integral
- presentation holds the code review and theory presentation docs; it also holds the "working paper" form of the theory documentation
- timing assesses how long it takes to solve KrusellSmith and OneAssetHANK, with comparisons against Matlab times.

Timing KrusellSmith

Code Section	Julia	Matlab
All, Do Checks	239 ms	234 ms
All, No Checks	249 ms	—
steadystate!	57.89 ms	130 ms
eqcond	164.427 ms	54.9 ms
reduction	11.43 ms	33.525 ms
gensys	646μs	9.09 ms
gensys!	5.53 ms	—

Do Checks: ensure mandatory settings defined, Γ_0 is identity, so not checking should be faster, right?

eqcond is slower for some reason. Automatic differentiation was tested to be faster earlier though, right?

gensys is faster when I don't use in-place Schur factorization. Ideas?

Timing OneAssetHANK

Code Section	Julia	Matlab
All, Do Checks	1.097 s	791.2 ms
All, No Checks	1.101 s	—
steadystate!	547.767 ms	537.8 ms
eqcond	414.913 ms	240.07 ms
reduction	130.938 ms	55.002 ms
gensys	2.503 ms	11.852 ms
gensys!	2.606 ms	—

eqcond is substantially slower again

Reduction here is substantially slower, too

gensys! is still slower...

CTDSGE

- New Github repository as part of the DSGE package for estimating a Brunnermeier-Sannikov style model
- Built with the same basic github repository, although it is very sparse currently and has no CTDSGE.jl file
- Model files just have the Julia translation of Matlab files no generalization made yet. Also probably slow.
- Will likely need different structure for a model object, e.g. no steady state, no canonical form matrices
- Equilibrium problem: solve a first-order PDE Basic finite difference scheme for approximating derivatives and Newton's method for solving resulting nonlinear system of equations.